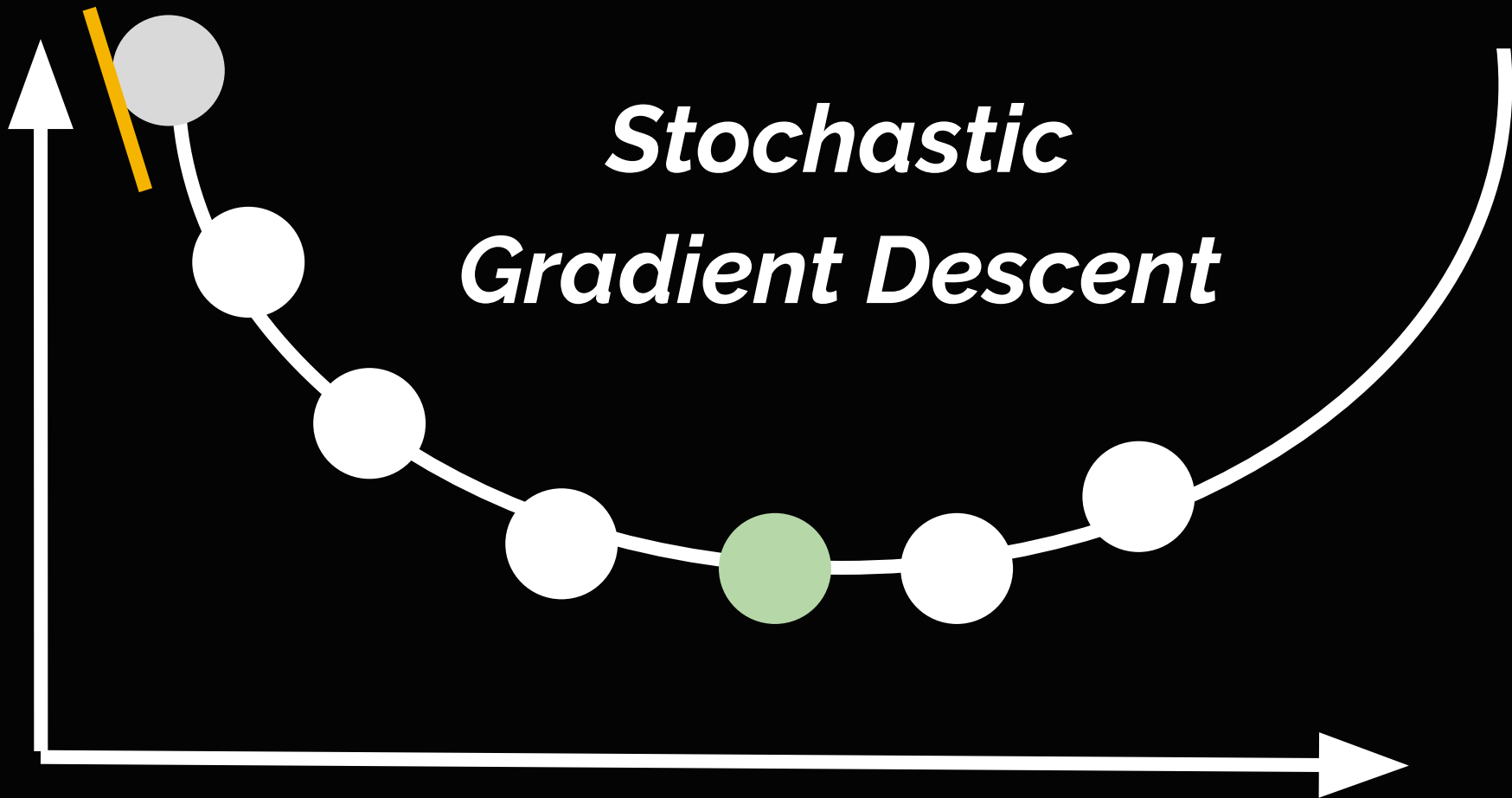
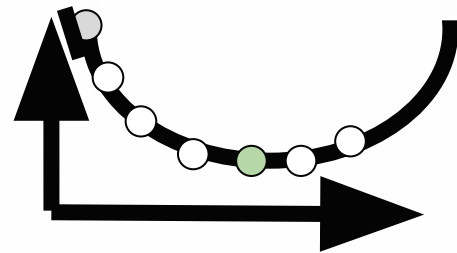
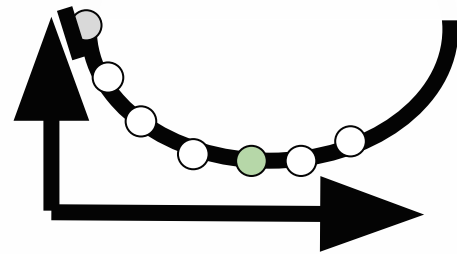


*Stochastic
Gradient Descent*

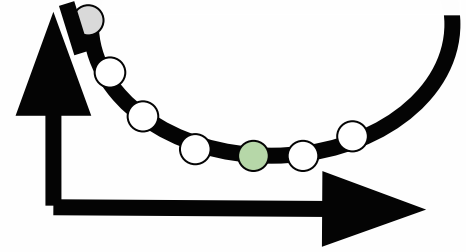




Gradient Descent

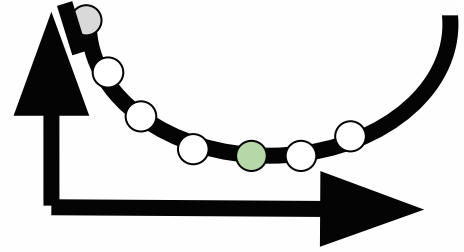


fundamental element in machine
learning algorithms

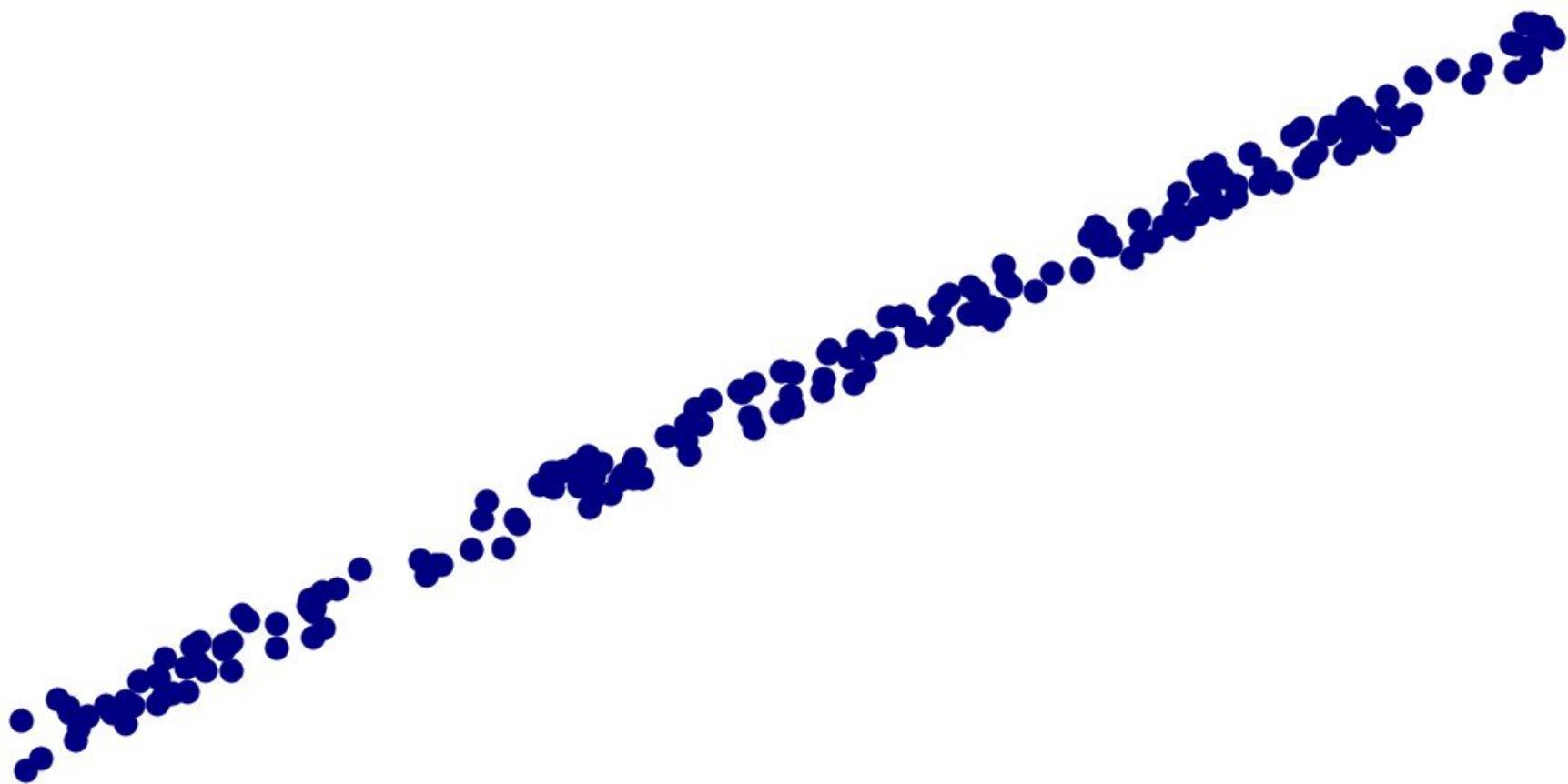


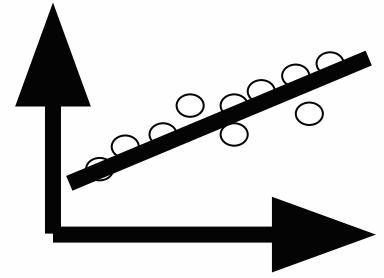
Update the parameters of a model

- to optimize the model



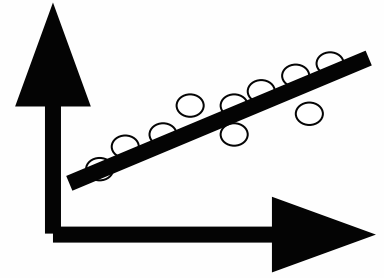
model updates parameters on its own!





Linear regression

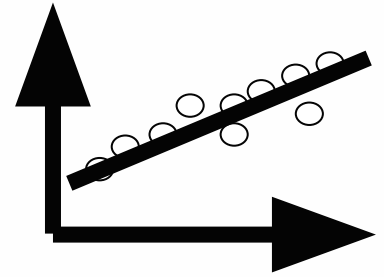
- model needs to find the **best fit line**
between the data points



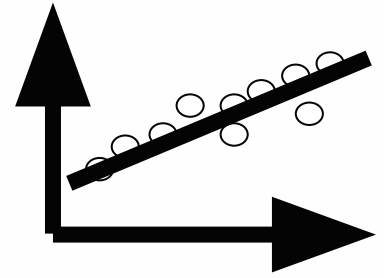
Linear regression

- $y = m * x + b$

$$y = m * x + b$$

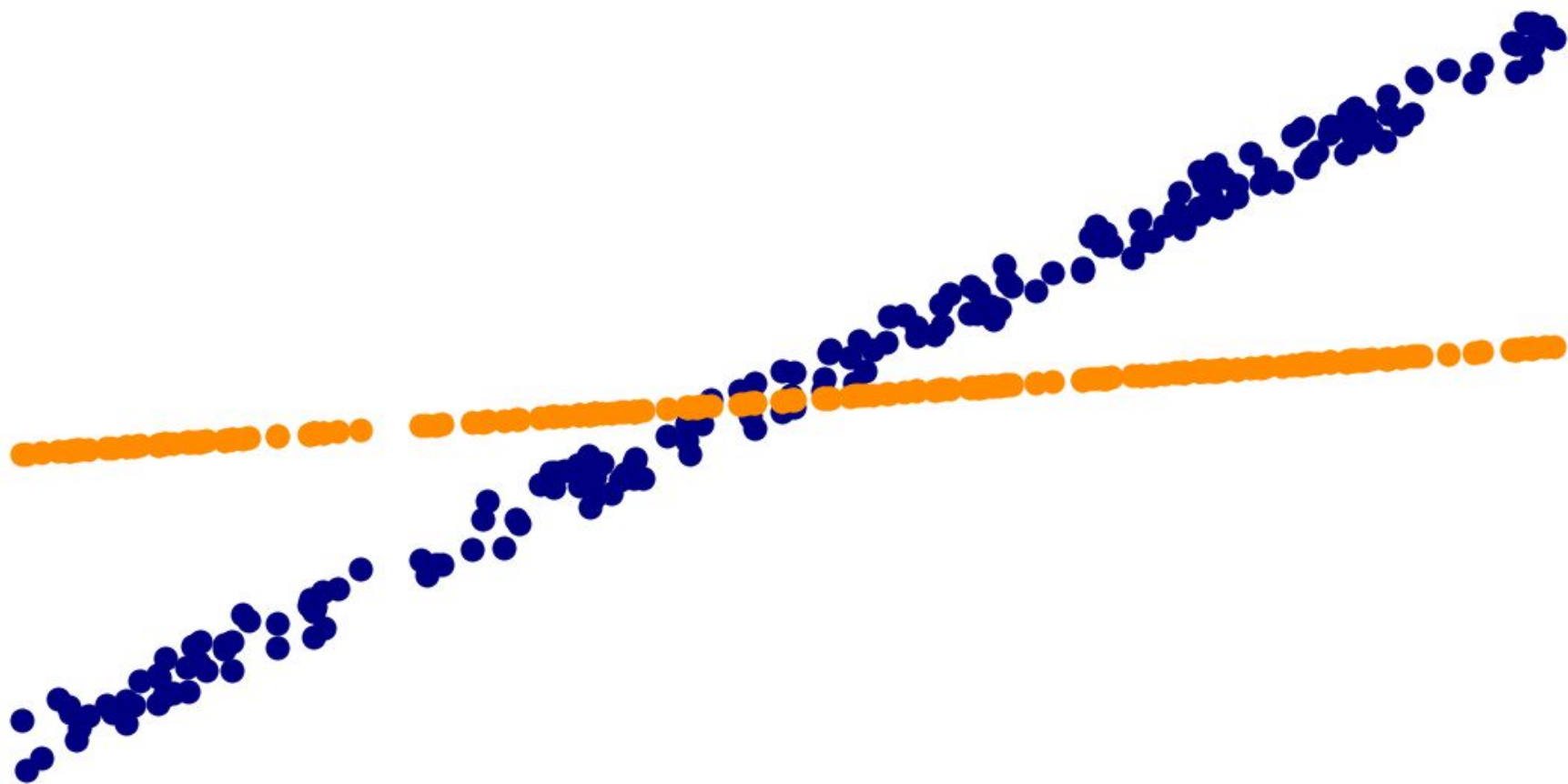


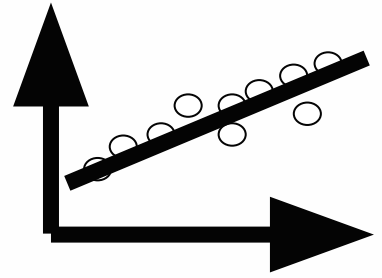
- We have x and y
- Model needs to determine **m and b**
 - learnable parameters in the model



Start with a random weight and bias

```
def predict(X, weight, bias):  
  
    y = weight * X + bias  
  
    return y
```





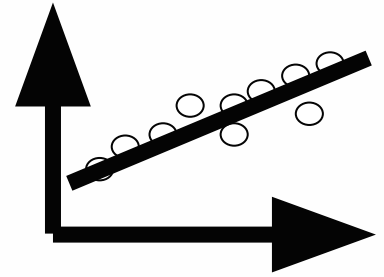
We need to measure the *quality* of predict()

- Use a **loss function**

loss function

- measures **how far** a model's predictions are from the actual labels
- determines **how good or bad** the model is.

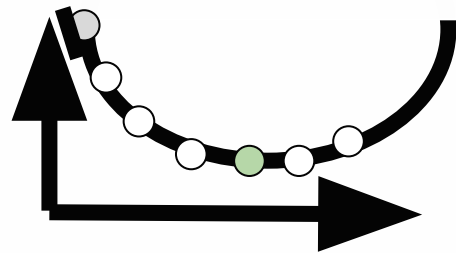
Linear regression



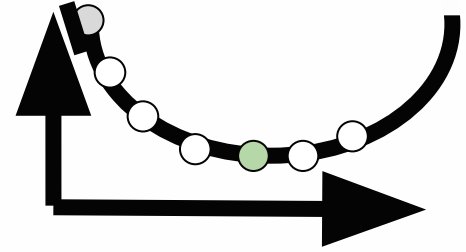
- Popular **loss function**
 - mean squared error (MSE).

```
def loss(X, y, weight, bias):  
  
    prediction = predict(X, weight, bias)  
  
    return numpy.mean(prediction - y) ** 2)
```


Loss function output

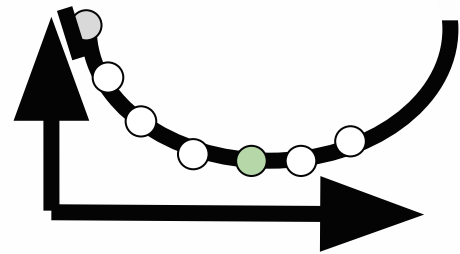


- The smaller the value, the better the model
 - only few errors were made in this case.
- A high value implies a worse model

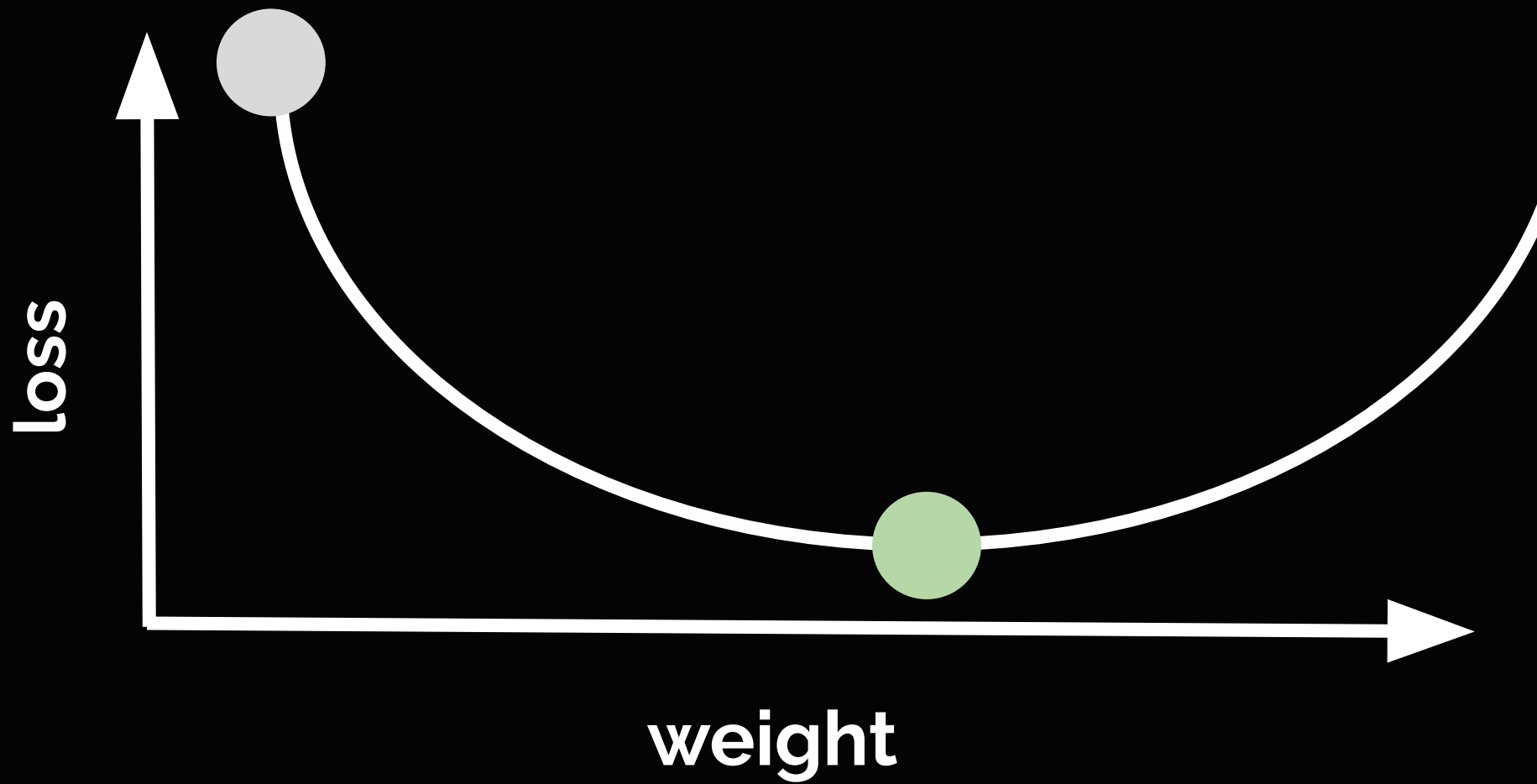


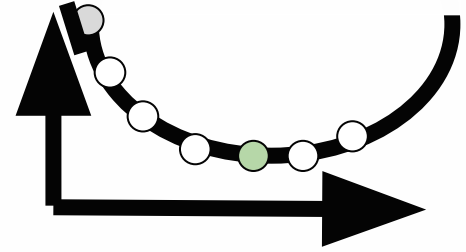
Goal

- To minimize loss function output



- We can measure how good or bad the model is
- Remember, we are trying to find values for weight and bias!

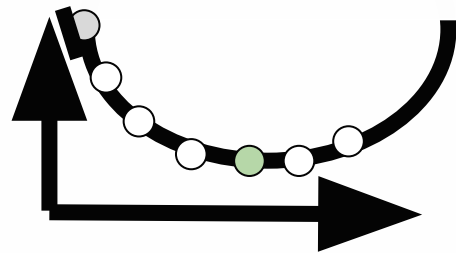




Gradient Descent

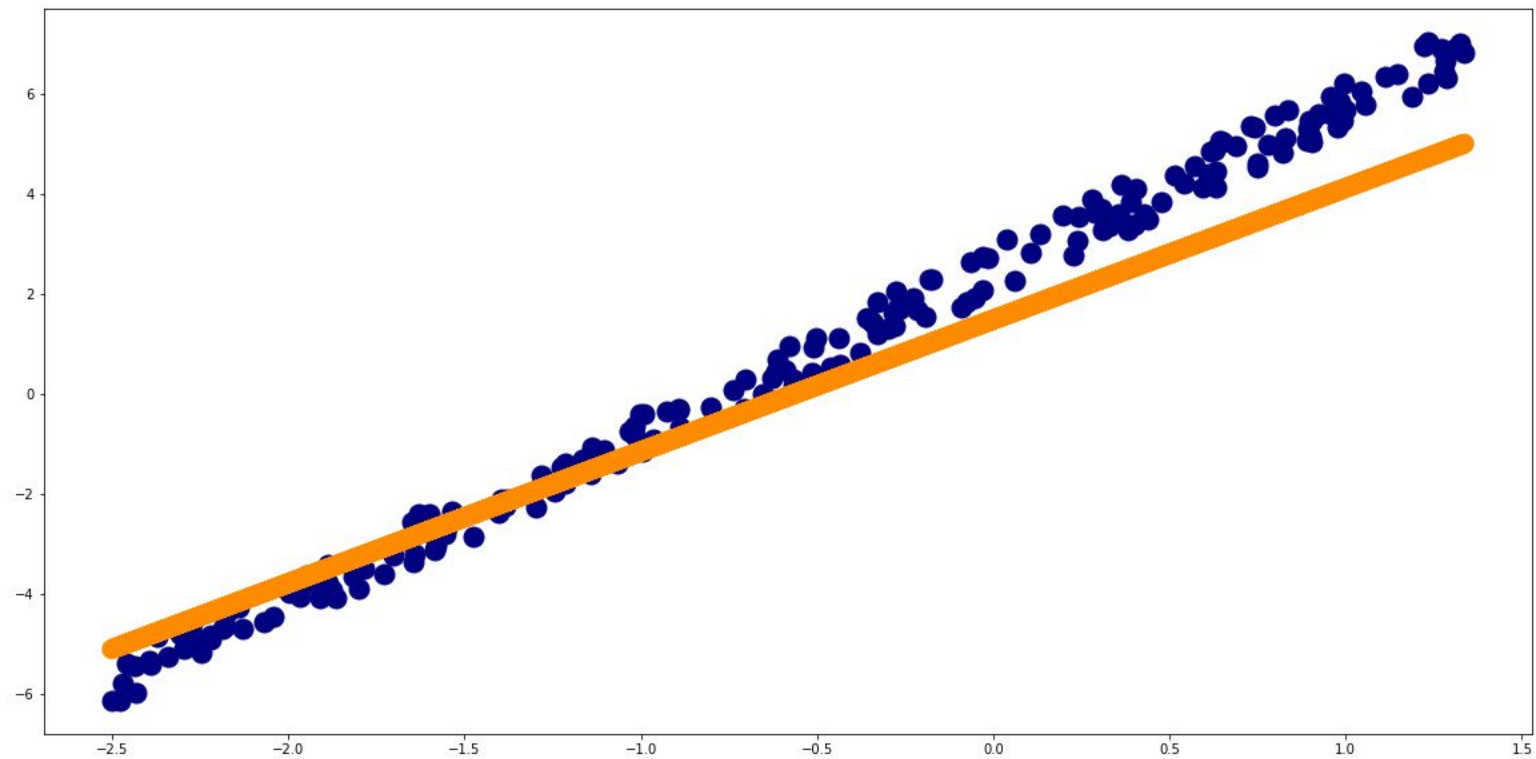
- find the **minimum** of our loss function
 - at the min, model makes the best predictions.

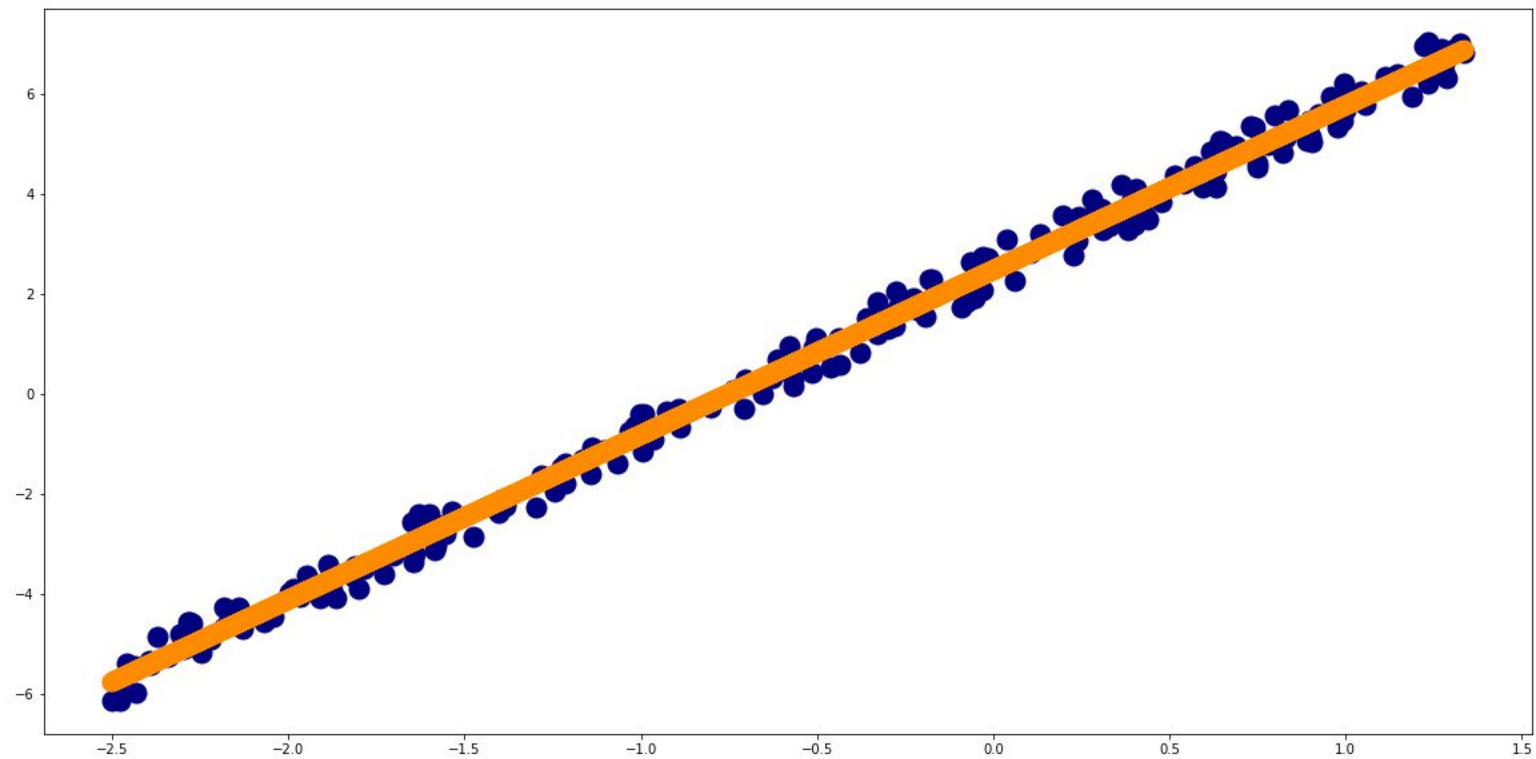
Gradient Descent



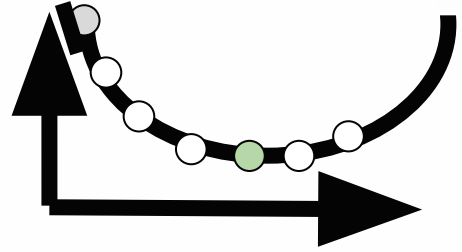
Goal: Minimize the loss

- Starts with parameters
- Moves towards new parameters to minimize the function
- Take steps in the negative direction of gradient function



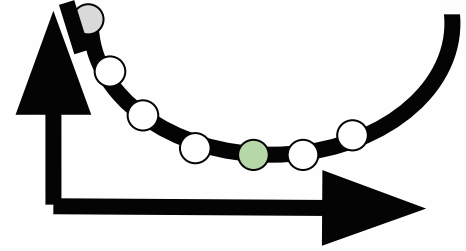


Problem



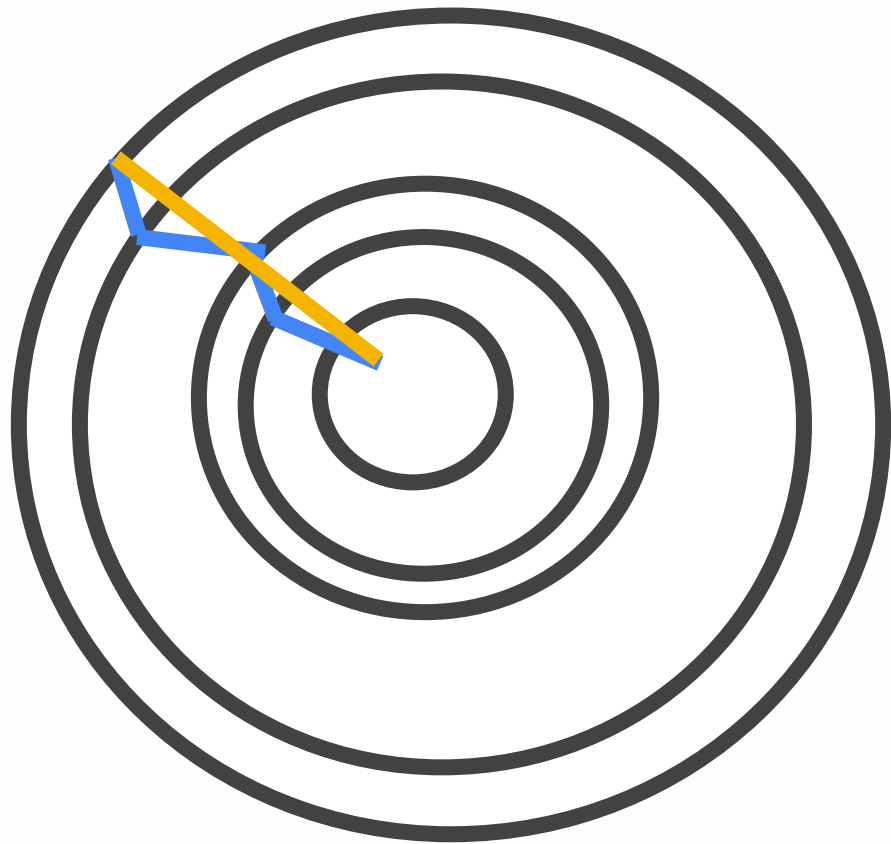
- In 1 iteration, you must **run through ALL samples** in your training set to do 1 parameter update

Solution

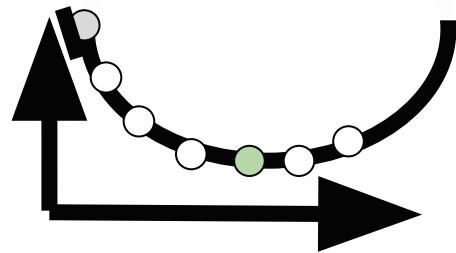


- **Stochastic Gradient Descent**

- use ONLY ONE or SUBSET of training
sample

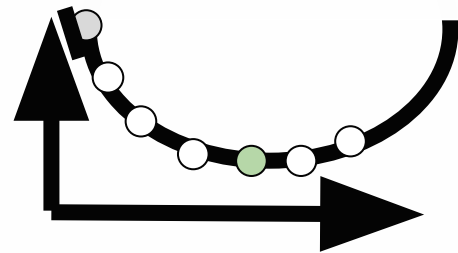


SGD often **converges much faster**



- but the error function is **not as well minimized**

Is that okay?



Usually, the close approximation of parameter values from SGD are enough

- they reach optimal values and oscillate there

Stochastic Gradient Descent

